システムLSI 検証技術入門

デバイスの記事

第15回

フォーマル・ベリフィケーション・ツールの実際

——ビヘイビア記述とRTL記述の間の等価性検証

藤田昌宏

ここでは，おもにビヘイビア記述とRTL記述の間の等価性検証について解説する．untimedのビヘイビア記述には時刻の概念がない．そのため，比較する入出力のタイミングを設計者が検証ツールに知らせてやる必要がある．パイプラインのような繰り返しの処理を行っている場合は，レイテンシとスループットを指定する．この指定には，数値や記述中の文の実行間隔などが用いられる．（編集部）

前回までの数回に渡り，プロパティ検証（プロパティ・チェッキング）について解説してきました注．今回からは，本連載のまとめという意味も含めて，現状の形式的検証技術やツールを実際に利用する際に，「どういう点が問題となるのか」，「どういうことに配慮しないとうまく使えないのか」といった問題について解説していきたいと思います．また，あわせて，それらのツールの近い将来の動向についても触れたいと思います．最近，ビヘイビア合成ツール（高位合成ツール）の導入も進みつつあることから，今回と次回はRTL（register transfer level），およびそれより上位の設計記述の間の等価性検証について考えます．

RTL回路とゲート・レベル回路の等価性検証では，原則としてフリップフロップの対応がつく場合がほとんどです．そのため，組み合わせ回路の等価性検証技術が適用可能であり，現在では，かなり大規模な回路の論理的等価性を検証できるようになってきています．これに対して，RTL記述間やRTL記述とビヘイビア記述の間の等価性検証においては，フリップフロップの対応がとれない場合が多く（部分的な対応がとれるケースは多いが…），組み合わせ回路ではなく順序回路の等価性検証技術を利用する必要があります．

ここでは順序回路の等価性の定義も含めて，現状の技術を解説していきます．

● 組み合わせ回路の等価性検証vs.順序回路の等価性検証

さて，まず，組み合わせ回路の等価性検証と順序回路の等価性検証の違いを確認しておきましょう．組み合わせ回路では，入力値が決定すると出力値は一意に決まります．したがって，過去の履歴を考えることなく，現在の入力だけを考えて等価性を調べればよいことになります．一方，順序回路では，内部にフリップフロップなどで実現した内部状態が存在し，出力値は，入力値とその内部状態の値で決定されます．一般に，内部状態を表す変数（論理変数）がn個あると，内部状態は最大2^n状態存在し，等価性問題の難易度は，規模とともに急激に増大します．

実際の検証アルゴリズムの概要を**リスト1**に示します．まず，初期状態どうしについて二つの設計を比較し，出力が等価であることを確認します．続いて，初期状態から到達可能な状態のすべてに対して，出力が等価であるか否かを検証していきます．この到達可能な状態の数が内部状態の数に対して指数関数的に増えていくため，状態を一つ一つたどりながら検証していくことは，事実上不可能になり

注：プロパティ検証については，本連載の第9回（2004年8月号，pp.135-141），第10回（2004年10月号，pp.136-141），第11回（2005年2月号，pp.144-148），第12回（2005年7月号，pp.130-136），第13回（2005年9月号，pp.144-149），第14回（2005年11月号，pp.124-129）で解説した．

KeyWord　フォーマル・ベリフィケーション，untimed，プロパティ，等価性検証，組み合わせ回路，順序回路，パイプライン，レイテンシ，スループット，SLEC，PSL

リスト1　対応する出力値を比較する時刻の指定
untimedのビヘイビア記述とRTL記述の等価性を検証することを想定している

```
Input In1;
Output Out1;
A = getInput(In1);
B = getInput(In1);
C = 3 * A;
D = C + B;
Out1 = putOutput(D);
```

（a）untimedのビヘイビア記述

```
Input In1;
Output Out1;
A = getInput(In1);        //時刻1（最初の時刻）
next; //時刻を進める
B = getInput(In1);        //時刻2
next  //時刻を進める
C = 3 * A;
D = C + B;
Out1 = putOutput(D);      //時刻3（最後の時刻）
```

（b）等価性の指定のための時刻の導入

ます．そこで，以前にも説明したように，状態を一つずつたどるのではなく，状態の集合を一度に処理するシンボリックなアルゴリズムが用いられています．

しかし，こうしたアルゴリズムを適用しても扱えるのはせいぜい数千ゲート規模の回路が限界です．つまり，順序回路の等価性検証が可能なのはLSIチップ全体の比較ではなく，回路のブロック単位の比較となります．したがって，ツールを利用する設計者もそれを念頭に置いて適用することが重要です．つまり，LSI設計の各ブロック単位で設計を進めていく際に，各ブロックの検証を十分に行ってからLSI全体の検証へと進むような設計手法を徹底することが必要になるわけです．

これに対して組み合わせ回路の等価性検証，とくに二つの回路がネットリストとしてかなり類似している場合（例えばタイミングを調整するために，回路のごく一部だけを変更した場合など）は，1000万ゲート規模の設計であっても，数時間程度で等価性を検証できます．

順序回路の等価性検証と組み合わせ回路の等価性検証では，このように性能（処理にかかる時間）に大きな差があるという認識を持つことが重要です．また，後で述べるように，順序回路の到達可能な状態をすべてたどるには，多くの処理ステップが必要となり，回路が少し大きくなると，実際には計算できなくなる場合がほとんどです．そこで，初期状態からではなく，特定の状態（あるいは，任意の状態）から一定ステップ（RTLの場合はクロック・サイクル数）の間，二つの回路が同じように動作するか否かだけを調べることも行われています．このようにすると，扱える回路の規模はかなり大きくなります．ただし，チェックしている等価性が，設計者の考える等価性と少し異なる意味になるので，注意が必要です．このことについても，後で述べたいと思います．

●「設計の等価性」を定義するには時刻の導入が必要

順序回路の等価性検証では，もう一つ，少しややこしい問題があります．それは，「二つの設計が等価」とはどういうことか，ということです．

「二つの設計が等価」であることを定義するなんて簡単なことだ，と思われるかもしれませんが，順序回路の出力は時間軸上で時々刻々と変化します．どのタイミングで等価であれば設計全体が等価であると言えるのかを明確に定義して，等価性検証ツールに知らせてやる必要があるのです．例えば，検証の対象が，「入力が決まると一定の計算を行い，出力が求まったらそれを出力する」という動作を1度だけ行う回路であれば，等価性は簡単に定義できます．しかし実際には，このようなハードウェアはほとんどなく，「入力から出力を計算する」という操作が何度も繰り返されます．そのため，比較している二つの設計に対して，どのタイミングの出力がどのタイミングの入力からの計算結果になっているかの対応をとらないと，等価性を検証できないことになります．この対応を自動的にとれるケースはあまりなく，設計者が明示的に指定しなければならない場合がほとんどです．そのためには，いくつかのくふうが必要となってきます．

例えばSpecCやSystemCなどのシステム・レベル言語を使って，ビヘイビア記述がいわゆるuntimedで記述されていたものとします．この設計記述そのものには，時刻の概念がありません．各文を順次実行するという意味で実行順序は決まっていますが，具体的にそれぞれの文をいつ実行するのかは未定となっています．一方，比較する相手のRTL記述にはクロックがありますから，明確な時刻が存在します．この両者に対して，入力が与えられるタイミングと出力が得られるタイミングを設計者が明示する必要があ

るのです．つまり，untimedの記述に対して何らかの時刻の概念を便宜的に導入することになります．

通常は，untimedの記述では計算時間がかからない(VHDLのシミュレーションのデルタ時間と同じ意味)と考えられるので，入力が来ると一瞬で出力が計算されることになります．しかし，これでは，入出力のタイミングを設計者が指定することはできません．設計対象が入出力を行うなど，外部と何らかのやり取りを行ったときに時刻が一つ進む，と考えるようにします．このように時刻の概念を導入することで，untimedの記述であっても，等価性を判定するべき時刻の指定を行えるようになります．

● ビヘイビア記述の動作に影響を与えないように導入する

簡単な例で説明しましょう．**リスト1(a)**に示すuntimedのビヘイビア記述を考えます．ここで，`In1`は入力端子，`Out1`は出力端子です．また，変数A，B，C，Dはビヘイビア記述で使用する内部変数です．ビヘイビア記述そのものは，入力端子`in1`から順に入力を取り込み，内部変数を利用して，`3*`(最初の入力値)＋(次の入力値)という計算を行い，出力端子`Out1`に出力しています．

このビヘイビア記述と，ビヘイビア合成などの処理から得られたRTL記述の間の等価性を検証しようとすると，RTL記述中の最初の入力を取り込むタイミングと，次の入力を取り込むタイミング，そして，計算結果を出力するタイミングの対応をとって，それを等価性検証ツールに対して明示的に示す必要があります．RTL記述のほうはクロックで時刻が決まるのでそれを調べればよいのですが，untimedのビヘイビア記述の場合，時刻の概念がないので，**リスト1(b)**のように時刻を導入することになります．最初の時刻を‘1’とし，`next`という文によって次の時刻に進めています．結果として，時刻1で最初の入力を受け取り，時刻2で次の入力を受け，計算を即座に行って，時刻3で出力すると指定されていることになります．

ここで注意しなければならないことがあります．時刻の導入はビヘイビア合成におけるスケジューリングに相当するわけですが，その目的は，あくまでもRTL記述との間で等価性を比較するべき時刻を指定するための手段であるということです．つまり，ビヘイビア記述の動作そのものに影響を及ぼさないように，時刻を導入しなければなりません．また，この時刻の指定法は，等価性検証ツールが理解可能でないといけません．通常は，`next`などの文を利用することで時刻を進めたり，指定された文を実行する時刻を指定したりできるようになっているか，あるいは時刻を進める規則があらかじめ決められており，それに従って設計者が時刻を指定するという方法が使われています．

実は**リスト1**の場合，入力値から一定の計算を行って出力値を求めるだけなので，わざわざこのように時刻を明示しなくても，「計算が終了したときの出力値を比較する」というように単純に扱うこともできます．しかし，このような計算を何度も行う**リスト2(a)**の場合はどうでしょう？ `while(1)`により無限に処理が繰り返されています．実際のハードウェアや組み込みシステムの場合，決して停止することなく処理を繰り返すのが普通です(これを，一般にreactiveシステムと呼ぶ)．

このようにループによる処理の繰り返しがあると，明示的に時刻を導入しなければ，等価であるかどうかを調べるタイミングがわからないことになってしまいます．繰り返しがある場合は，**リスト2(b)**のように，各繰り返し実行のそれぞれについて，相対的な時刻で入力と出力のタイミングを指定することになります．このとき，各入力は一定

リスト2　ビヘイビア記述が同じ計算を繰り返す場合の時刻の指定
(b)では，ループの繰り返しごとに相対時間を指定している

```
Input In1;
Output Out1;
while(1){
  A = getInput(In1);
  B = getInput(In1);
  C = 3 * A;
  D = C + B;
Out1 = putOutput(D);
}
```

(a) untimedのビヘイビア記述

```
Input In1;
Output Out1;
While(1) {
  A = getInput(In1);     //時刻 i
  next;   //時刻を進める
  B = getInput(In1);     //時刻 i+1
  next   //時刻を進める
  C = 3 * A;
  D = C + B;
  Out1 = putOutput(D);   //時刻 i+2
}
```

(b) 等価性の指定のための時刻の導入

周期で入ってくるので，周期も指定する必要があります．詳細については，パイプライン処理の場合といっしょに説明しましょう．

● パイプライン処理ではレイテンシとスループットに注目

ビヘイビア記述では各処理を順々に実行するように書かれている設計であっても，RTLになると，性能を向上させるために処理を並列化したり，パイプライン構成を導入したりすることが頻繁に行われています．パイプライン処理を行う場合，最初のデータの処理が終了する前に次のデータの処理が開始されることになり，結果的に出力も次々と得られることになります．このようなパイプライン処理も含めて，繰り返しのある処理の等価性を調べるべきタイミングは，どのように指定すればよいのでしょう？ この場合は，計算に必要な時間と入力を受け付ける頻度を指定することになります．

一般に，入力が入ってからその設計が出力を返すまでに必要な時間のことを「レイテンシ」と呼びます．例えば先ほどのビヘイビア記述の場合，計算は瞬時に終了する，つまりどのような計算（の組み合わせ）も1ステップで終了することになるので，レイテンシは1になります．これに対して，対応するRTL記述では，通常，いくつかのステップ（クロック・サイクル）が必要になり，そのステップ数を指定することになります．また，入力値によって，実行ステップ数が異なるような場合，それに応じて値を指定しなければならないことになります．このような場合，実用上，等価性検証の適用がかなり難しいケースも出てきます（これについては，後でまとめて説明する）．

また，一つのデータが入力された後，次のデータを入力するまでのステップ数のことを「スループット」と呼びます．パイプライン処理をまったく行っていない場合はレイテンシとスループットはほぼ一致するはずですが，パイプライン処理を行っていると，スループットをレイテンシよりも小さい値にできます．つまりこの場合，内部で複数データについて並列処理が行われているわけです．

スループットが一定の定数値になる場合がある一方で，パイプライン処理がデータの値によって一時的に中断したり，前のデータの処理の終了が延期されるといったことも，複雑なパイプライン処理ではよく起きます．典型的なのは，マイクロプロセッサなどのパイプライン処理です．通常は，各クロック・サイクルごとに命令を実行していきますが，

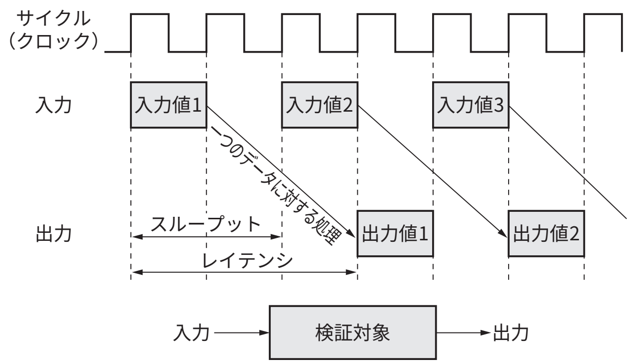


図1　レイテンシとスループット
一つのデータに対する計算時間が「レイテンシ」，入力データ間隔が「スループット」である

データ・ハザードなどが生じると，次の命令が現在の命令の実行結果を使っているため，次の命令の処理を待たせる必要があります．このような場合，スループットは一定値にはなりません．データ値に依存します．

以上をタイミング・チャートの形にまとめたものを**図1**に示します．レイテンシやスループットの意味と意図を読み取ってください．一つのデータに対する計算時間がレイテンシで，入力データ間隔がスループットです．なお，このレイテンシとスループットの指定は，比較している二つの設計記述のそれぞれに対して行います．もし，このレイテンシやスループットの値をまちがって指定すると，等価な設計でも「等価ではない」という検証結果が得られることになります．この場合，まちがった設計が正しいと判定される可能性は小さいのですが，理論的には起こりえます．設計の誤りかたが，まちがったレイテンシやスループットに偶然合致する場合にそうなります．偶然にそうなることは非常に少ないのかもしれませんが，可能性がゼロではないので注意が必要です．

● 数値の代わりに，記述中の文の実行間隔として指定

さて，上記の例で，レイテンシとスループットがともに一定値にならない場合はどのように指定すればよいのでしょう？ このような場合，一般には現在入力されるデータの値や以前のデータの値によって内部処理が異なり，結果としてレイテンシ，スループットともに変化します．このような値の変化を設計者が正確に把握し，その一種の“計算式”のようなものを指定しなければならない，ということになります．しかし，そのような計算式を正しく指定する

リスト3　入力値によってレイテンシやスループットが変化する設計例

```
Input In1;
Output Out1;
While(1) {
  X = Get(In1);
  Y = Get(In1);
  A = 0;
  B = X;
  While(B >= Y){
    B = B - Y;
    A = A + 1;
  }
  Out1 = A;
  Out1 = B;
}
```

(a) レイテンシやスループットがデータ値に依存する例

```
Input In1;
Output Out1;
While(1) {
  X = Get(In1);// InputTiming1
  Y = Get(In1);// InputTiming2
  A = 0;
  B = X;
  While(B >= Y){
    B = B - Y;
    A = A + 1;
  }
  Out1 = A;    // OutputTiming1
  Out1 = B;    // OutputTiming2
}
```

(b) レイテンシやスループットを設計記述の文で指定する方法

ことは，通常はかなり難しいと思われます．例えばビヘイビア合成ツールを利用して自動的にRTL記述を生成した場合，各クロックごとの動作はツールによって自動的に決められるわけですから，設計者はその詳細を把握していないことになります．

ですから一般的には，レイテンシやスループットを示す計算式を設計者がツールに与えられるとは限りません．しかしこれが与えられないと，等価性検証ツールはどのタイミングでどのように各出力の等価性を調べたらよいのかを判断できなくなってしまいます．そこで，レイテンシやスループットを示す方法として，設計記述を利用する方法が提案されています．

これは，レイテンシやスループットを数値として指定する代わりに，指定された記述中の文(一般には，入力や出力を行っている文になる場合が多い)の実行間隔として指定しようという考えかたです．設計記述において，外部端子から入力値を取り込んでいる文から，その値を用いた計算結果を出力している文までの間隔がレイテンシに相当するので，数値ではなくそれら二つの文を指定することでレイテンシを示そうというのです．また，一つの入力文から次のそれと同じ入力文を実行するまでの間隔はスループットに相当するので，その文を指定することにより，スループットを指定できることになります．

● 実行文がいつ実行されるかを知らなくてもOK

例を示しましょう．**リスト3(a)**を見てください．入力端子から二つの入力をX，Yとして受け取り，Xの値(設計記述ではXの値はBに代入されている)からYの値を何回引き算できるかを計算することで，「X割るY」を計算する記述です．

この例では，この割り算処理を入力データごとに計算することをwhile(1)により無限に繰り返しています．このような場合，whileループを回る回数は入力データに依存します．つまり，レイテンシやスループットは定数にはならず，入力データの値によります．この例の場合，その値は割り算の商の値と一致します．つまり，レイテンシやスループットの値を計算式で指定するにしても，そこには割り算が現れることになります．これは，検証対象の計算と同じ複雑さの式がレイテンシやスループットの指定を行う計算式に現れることになり，非常に複雑なことを指定していることになります．等価性検証の処理も簡単ではありません．

そこで，**リスト3(b)**のように，設計記述中の文で入出力を行っているものに，一種の“名まえ”を付けます(**リスト3(b)**では，InputTiming1，InputTiming2，OutputTiming1，OutputTiming2)．これらを使ってレイテンシやスループットを指定します．この例では，OutputTiming1 - InputTiming1やOutputTiming2 - InputTiming2がレイテンシ，next(InputTiming1) - InputTiming1やnext(InputTiming2) - InputTiming2がスループットとなり，このように指定することになります．ここでInputTiming1，InputTiming2，OutputTiming1，OutputTiming2は，それが書かれている設計記述の文が実行される時刻として定義されます．したがって，ビヘイビア・レベルのuntimedの記述でもRTL記述でも同じように指定できます．

この実行文の指定は，実行文がいつ実行されるかを設計

者が知らなくても,「その文を実行する時刻」ということで明示できるので問題ありません.その意味では,レイテンシやスループットを数値で指定するよりもわかりやすい方法であると言えます.ただし,動作が複雑になってくると,データを入力したり出力したりする文が1ヵ所ではなく複数ヵ所あることも珍しくなく,その対応の指定は相当煩雑になってしまうかもしれません.そのような場合は,この方法がかならずしも直感的でわかりやすいとは言えなくなってしまいます.

● READYやDONE,プロトコルの信号で指定する方法も

レイテンシやスループットを数値以外の形で指定する方法として,設計記述の入力受け付け可能状態を示す信号や,出力値が利用可能であることを示す信号を使う方法があります.例えば,「READYという信号が'1'になったら,入力データが入る」,「DONEという信号が'1'になったら,出力データが計算できている」ということで,レイテンシやスループットを示します.これは,READYやDONEといった信号が明確に定義されている設計ではきわめて有効な方法です.

また,入出力の値の受け渡しに対して一定のプロトコルが存在する場合(例えば,ハンドシェイクを行う場合)は,そのプロトコルの信号を利用してレイテンシやスループットを指定することも可能です.逆にそのような信号がない場合には,そのままでは利用することができません.そのようなときは,等価性検証のために,そのようなプロトコルに対応する設計記述を一時追加するなどのくふうが必要になります.ただし,そのような設計記述を追加するには,ある程度設計記述の内容を理解している必要があります.ビヘイビア合成で自動生成されたRTL記述に対してこの方法を適用することは,かならずしも容易ではありません.

● 次回は市販ツールの適用例を紹介

以上の議論を説明するために,次回は,市販の等価性検証ツールを用いて設計記述の検証を行った例を紹介します.現在,一般に利用可能な順序回路の等価性検証ツールとして,米国Calypto Design Systems社の「SLEC」と呼ばれるツールがあります.このSLECを使った等価性検証の結果を示したいと思います.これにより,順序回路の等価性検証がどのような場合に効果があり,どのような場合には厳密な検証が困難になるかを説明したいと思います.

なお,本連載で使用するツールはCalypto社のものですが,同じことは,モデル・チェッカやプロパティ・チェッカと呼ばれているそのほかのフォーマル・ベリフィケーション・ツールでも,「出力が等価である」というプロパティを与えることによって実現できます.プロパティ・チェッカにおいて,等価性に関するプロパティをPSL(Property Specification Language)などのアサーション言語を利用してうまく記述できれば,そのままプロパティ・チェッカを順序回路の等価性検証ツールとして利用できます.その意味では,順序回路の等価性検証ツールとは,現行のプロパティ・チェッカに,等価性の定義を行いやすくするユーザ・インターフェース(等価性指定のためのアサーション記述法)と,等価性検証であることを利用して処理を高速化した検証エンジン(例えば,二つの設計の内部の等価点を積極的に利用するなど)を追加したものとみなすこともできます.

次回はこうした点についても,もう少し詳しく説明します.さらに,順序回路の等価性検証に関する研究動向,すなわち近い将来利用可能と考えられる技術についても紹介したいと思います.

ふじた・まさひろ
東京大学 大規模集積システム設計教育研究センター(VDEC)教授

<筆者プロフィール>
藤田昌宏.1985年,東京大学大学院 情報工学専門課程修了.工学博士.同年,富士通研究所入社.論理設計用CAD技術の研究開発に従事.1993年より米国富士通研究所へ出向,CAD研究グループを立ち上げる.2000年3月より東京大学大学院 工学系研究科電子工学専攻 教授.2004年4月より東京大学 大規模集積システム設計教育研究センター(VDEC)教授.ディジタル・システム設計支援技術,とくにシステム・レベル合成,論理合成,論理検証を中心に研究している.